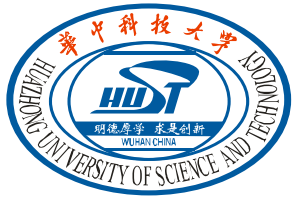
# DShuffle: DPU-Optimized Shuffle Framework for Large-scale Data Processing

**Chen Ding**[1], Sicen Li[1], Kai Lu[1], Ting Yao[2], Daohui Wang[2], Huatao Wu[2], Jiguang Wan[1], Zhihu Tan[1], Changsheng Xie[1]

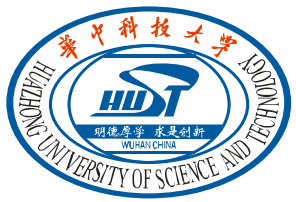
HUST[1], Huawei Cloud[2]




武汉光电国家研究中心
WUHAN NATIONAL LABORATORY FOR OPTOELECTRONICS


# Agenda

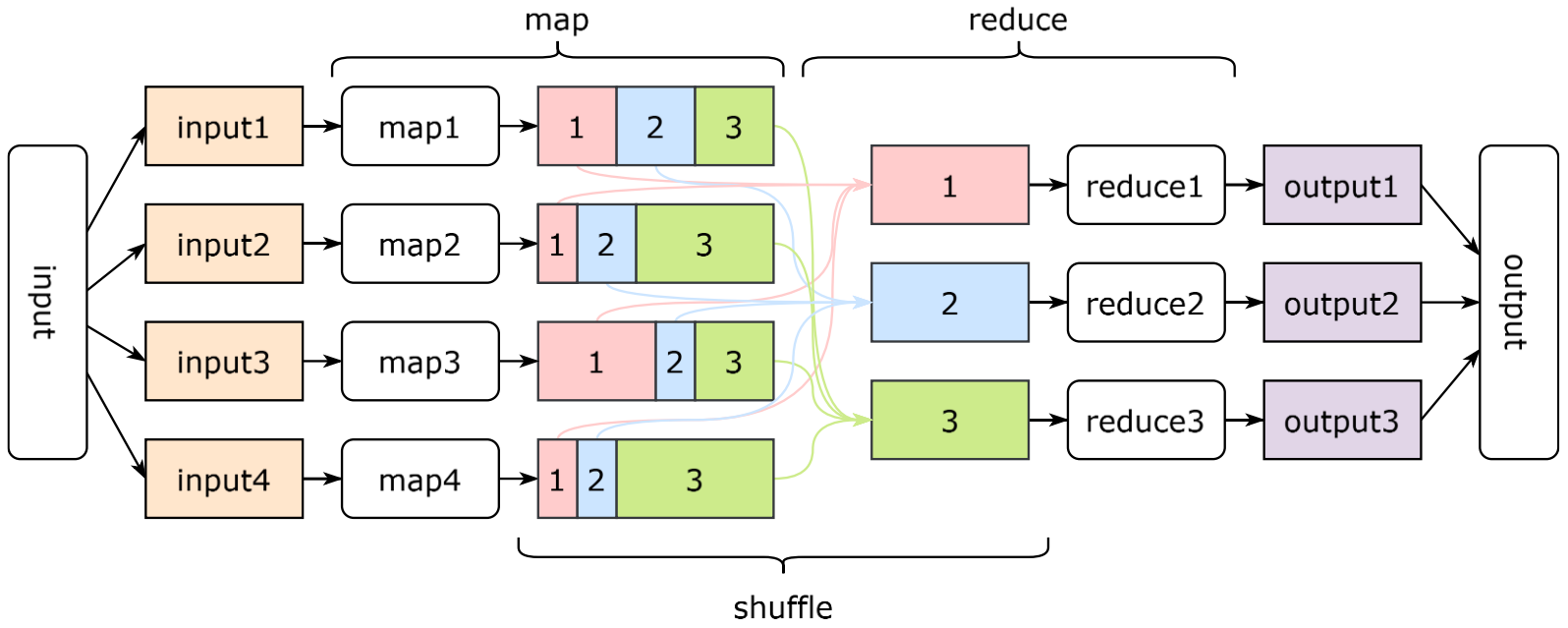
- Background & Motivation
- Design & Techniques
- Evaluation
- Conclusion




武汉光电国家研究中心
WUHAN NATIONAL LABORATORY FOR OPTOELECTRONICS

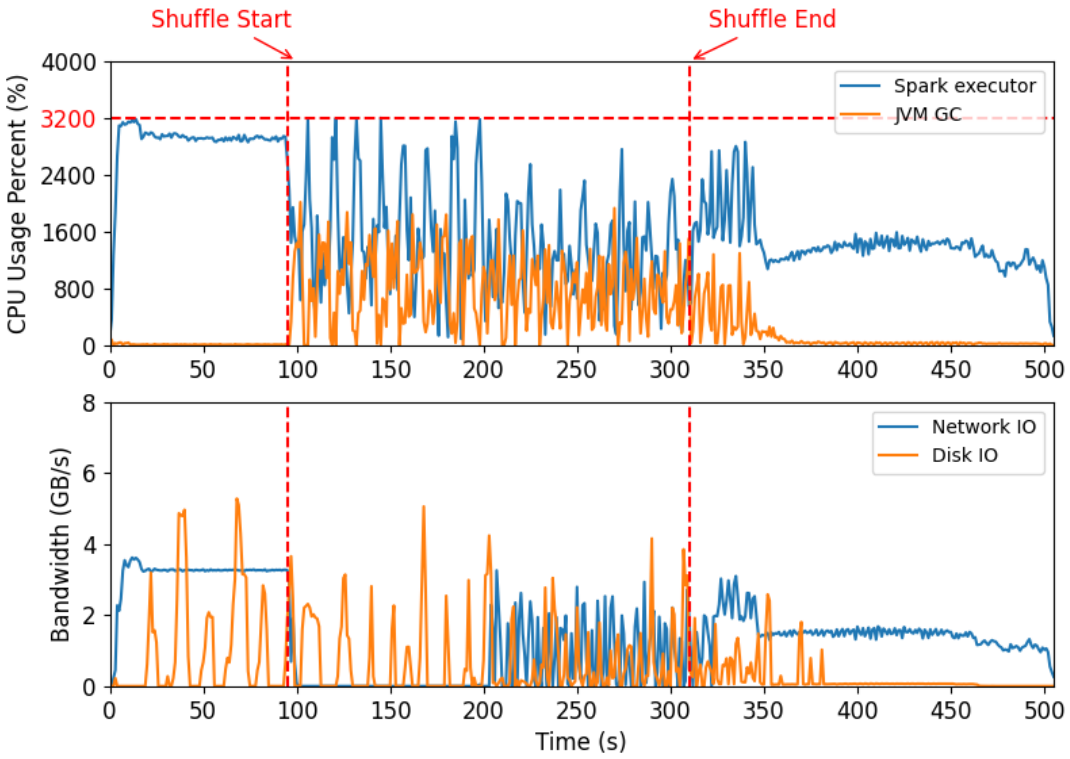

# Data shuffle in data-intensive applications

Data Shuffle is a fundamental operation in big data analysis and processing system.

# CPU overhead of data shuffle

For modern storage and network devices, data shuffle has significant CPU overhead.



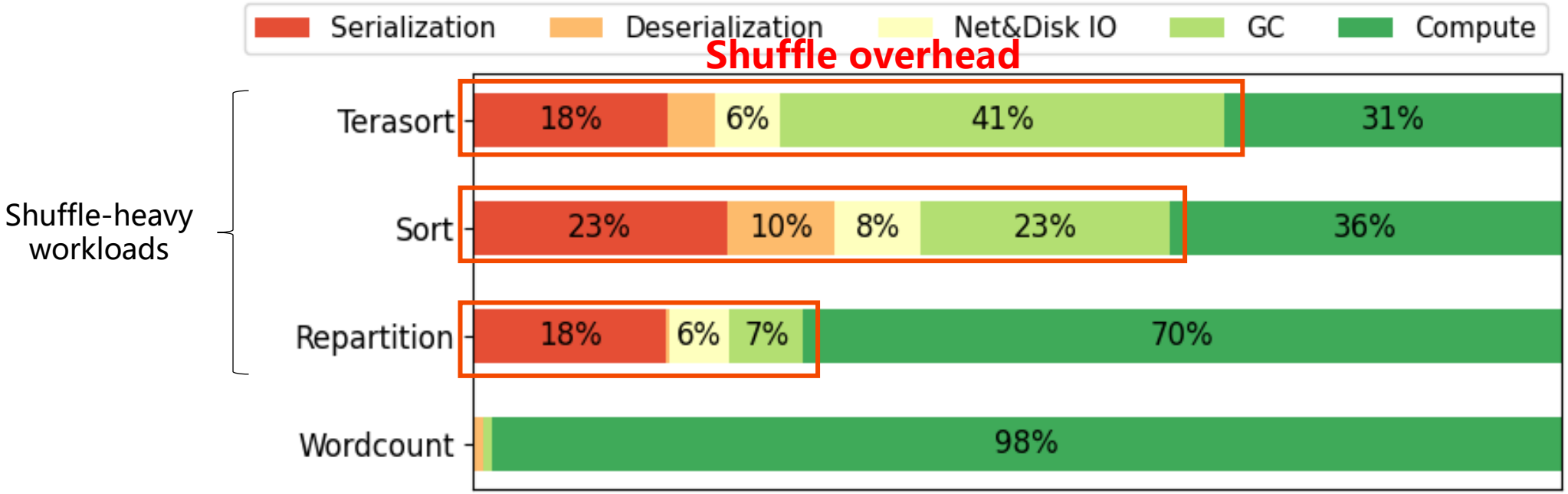

HiBench Sort workloads (285GB)

- Configurations
  - 2 * 16C32GB Compute Nodes
  - 100Gbps Network
  - 7GB/s PCIe 4.0 SSD

- Approximately **30** CPU cores are occupied during shuffle stage.
- Network and disk bandwidth utilization are only **25%** and **51%** respectively.

# CPU overhead of data shuffle

Substantial number of CPU cycles are spent on (de)serialization and garbage collection.

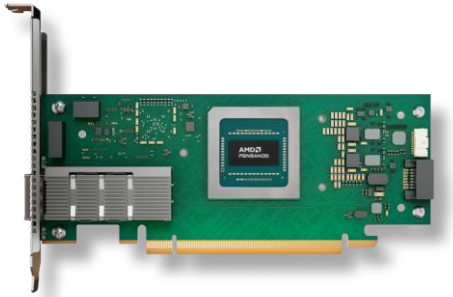


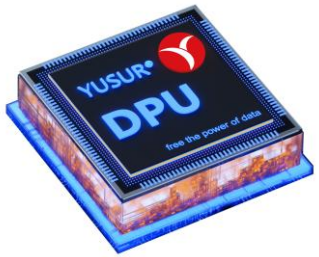

CPU Time Breakdown using Perf Tool

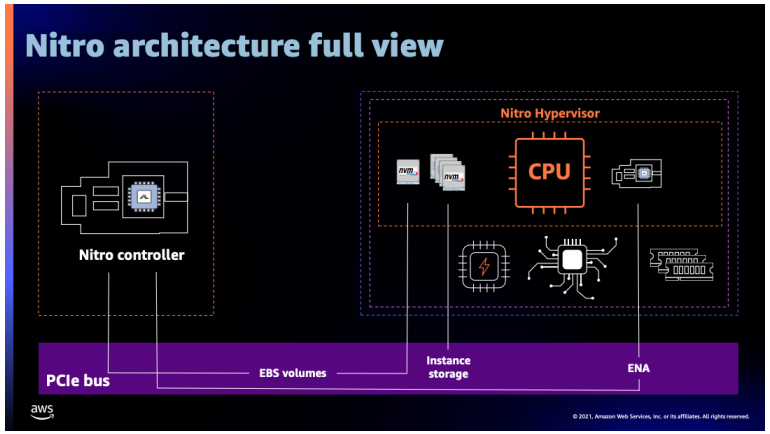
# DPU as An Opportunity

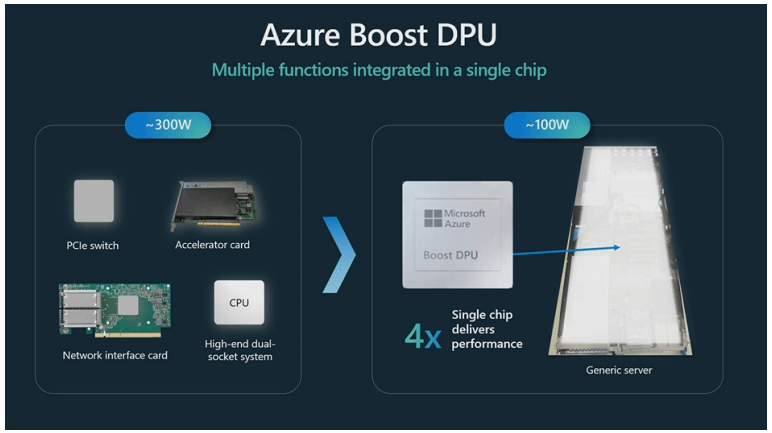
DPUs are widely deployed to reduce CPU overhead caused by storage and network tasks.

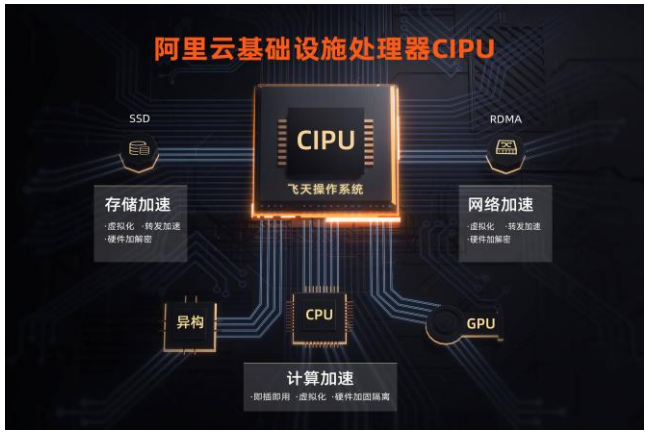




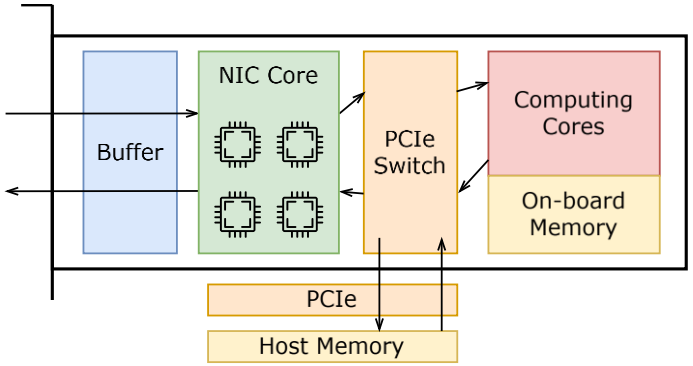



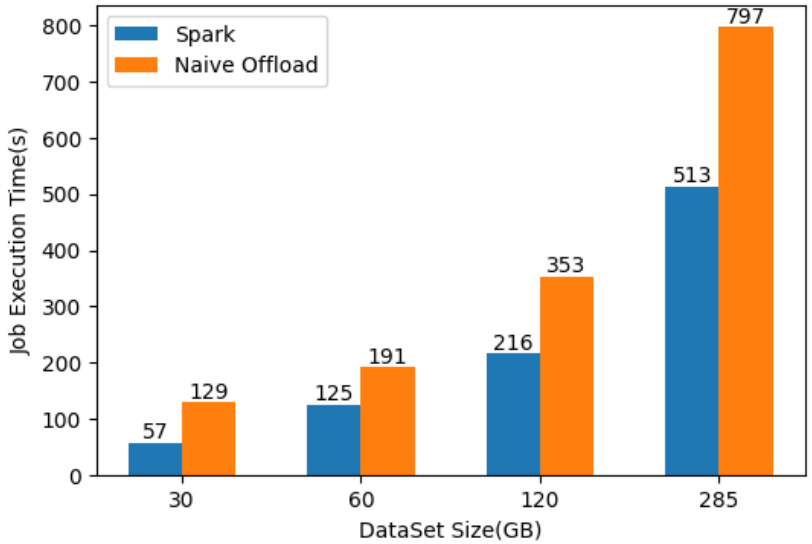



# DPU as An Opportunity

DPUs are well suited for offloading data shuffle operations, but there are challenges.



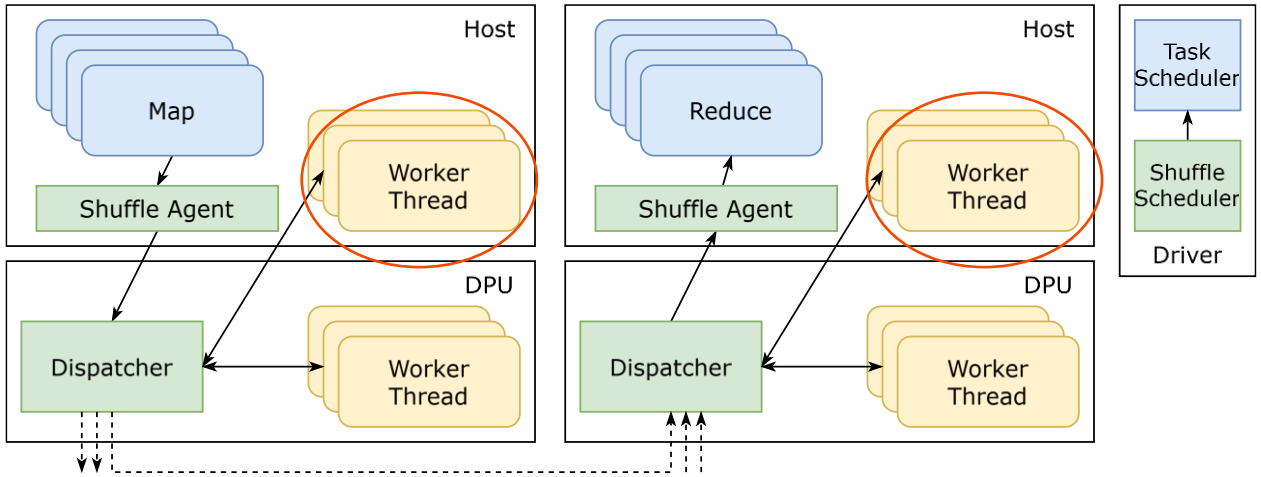



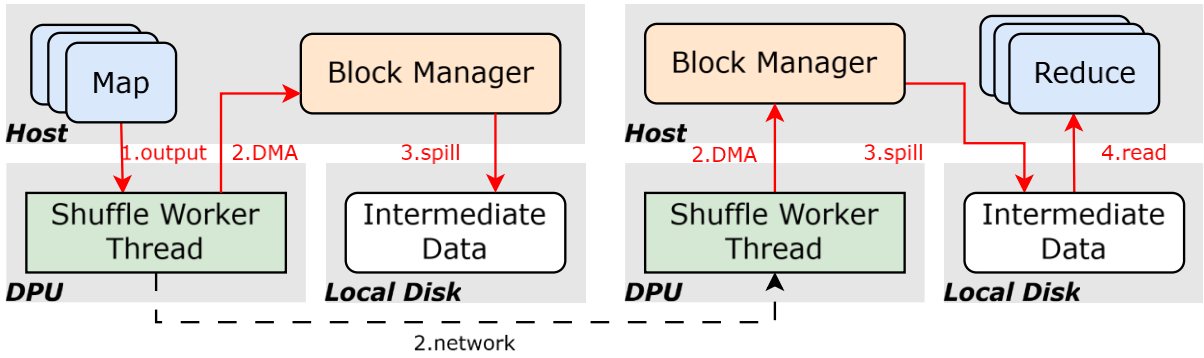

Task time increased 1.52x ~ 1.68x

- Opportunities
  - High memory access and IO bandwidth
  - High programmability
  - On the data path of data shuffle operations
- Challenges
  - The general-purpose cores have a low frequency and are limited in number

# Existing optimizations

Use dynamic offloading strategies to avoid DPU computation bottlenecks.

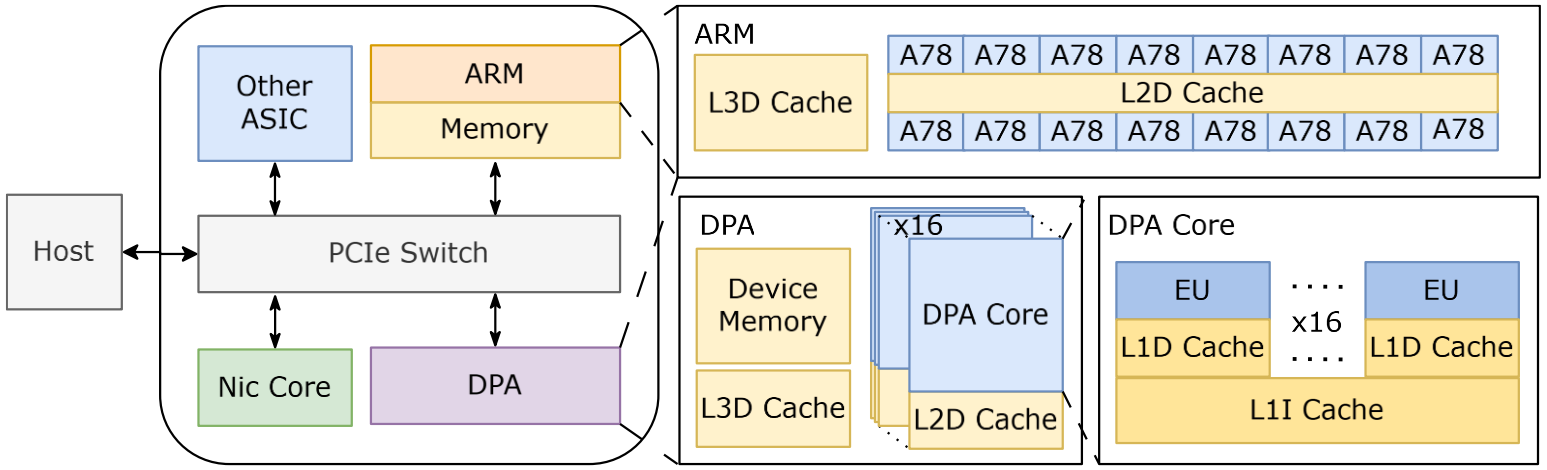



- Issues
  - Dynamic unloading still consumes host CPU resources.
  - Persisting intermediate results introduces additional data copies.
  - The (de)serialization overhead still exists.

# Our Key Insight

Modern DPUs can achieve complete shuffle offloading without sacrificing performance.




- Higher bandwidth RDMA NIC for data transfer
- Higher frequency and number of ARM cores for computation
- Data-path accelerator for serialization acceleration
- Direct storage access capabilities for data persistence

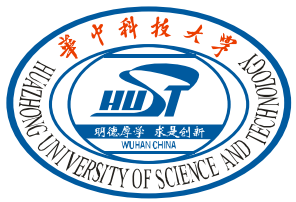
**BlueField-3 DPU**

| Components | Specifications |
|---|---|
| RDMA NIC | 2 * 200Gbps |
| General-purpose cores | **16 * 3.0 GHz** ARM Cortex A78 |
| Accelerators | **DPA**, DMA, NVMe-oF, … |
| DRAM | 32GB DDR5 Memory |

| | **DPA** | | **DPA** |
|---|---|---|---|
| CPU | RV64IMAC | L1D Cache | 1KB×256 |
| Core Number | 16 | L1I Cache | 1KB×16 |
| Thread Number | 256（190） | L2D Cache | 1.5MB×16 |
| Frequency | 1.8GHz | L3D Cache | 8MB×1 |
| **Access Arm Memory Bandwidth** | **~20GB/s** | **Device Memory** | **512MB** |
| **Access Host Memory Bandwidth** | **~40GB/s** | **Device Memory Bandwidth** | **~80GB/s** |
| Thread Schedule Latency | 3~7μs | Access Memory Latency | 800ns~1μs |

# Agenda

- Background & Motivation
- Design & Techniques
- Evaluation
- Conclusion

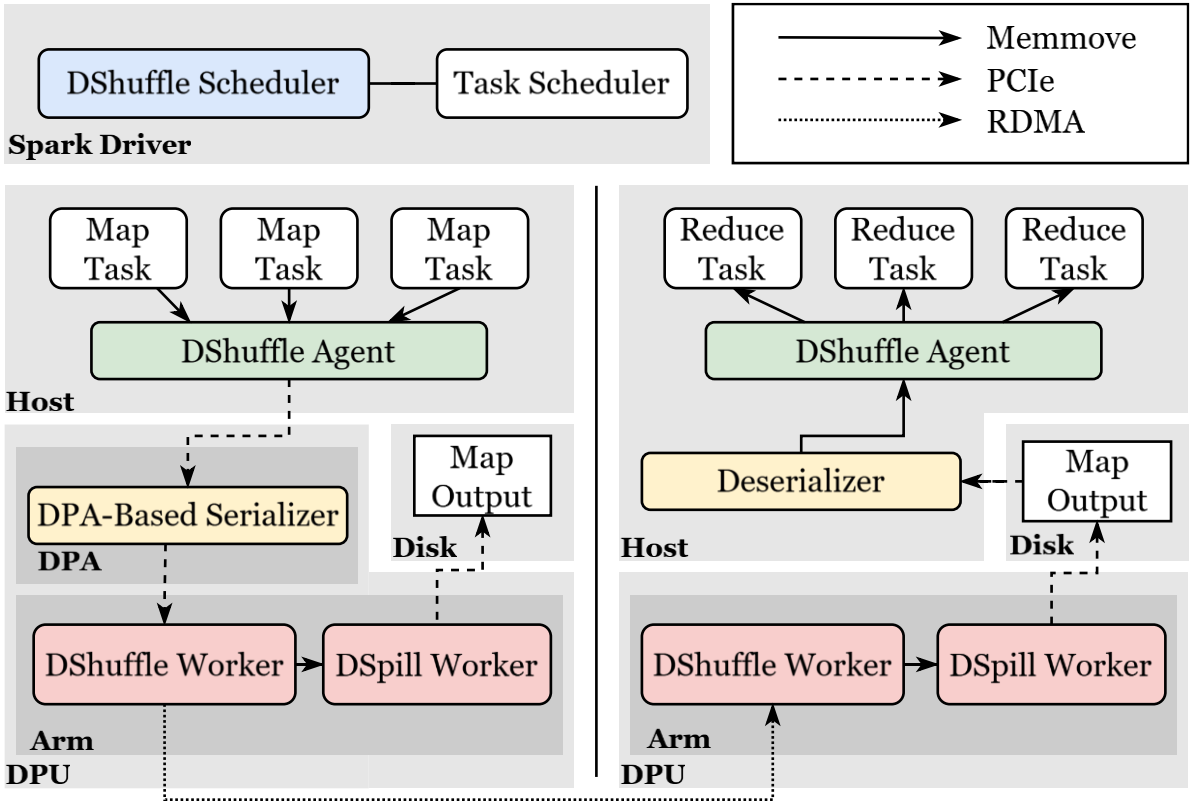




武汉光电国家研究中心
WUHAN NATIONAL LABORATORY FOR OPTOELECTRONICS


# DShuffle Overview

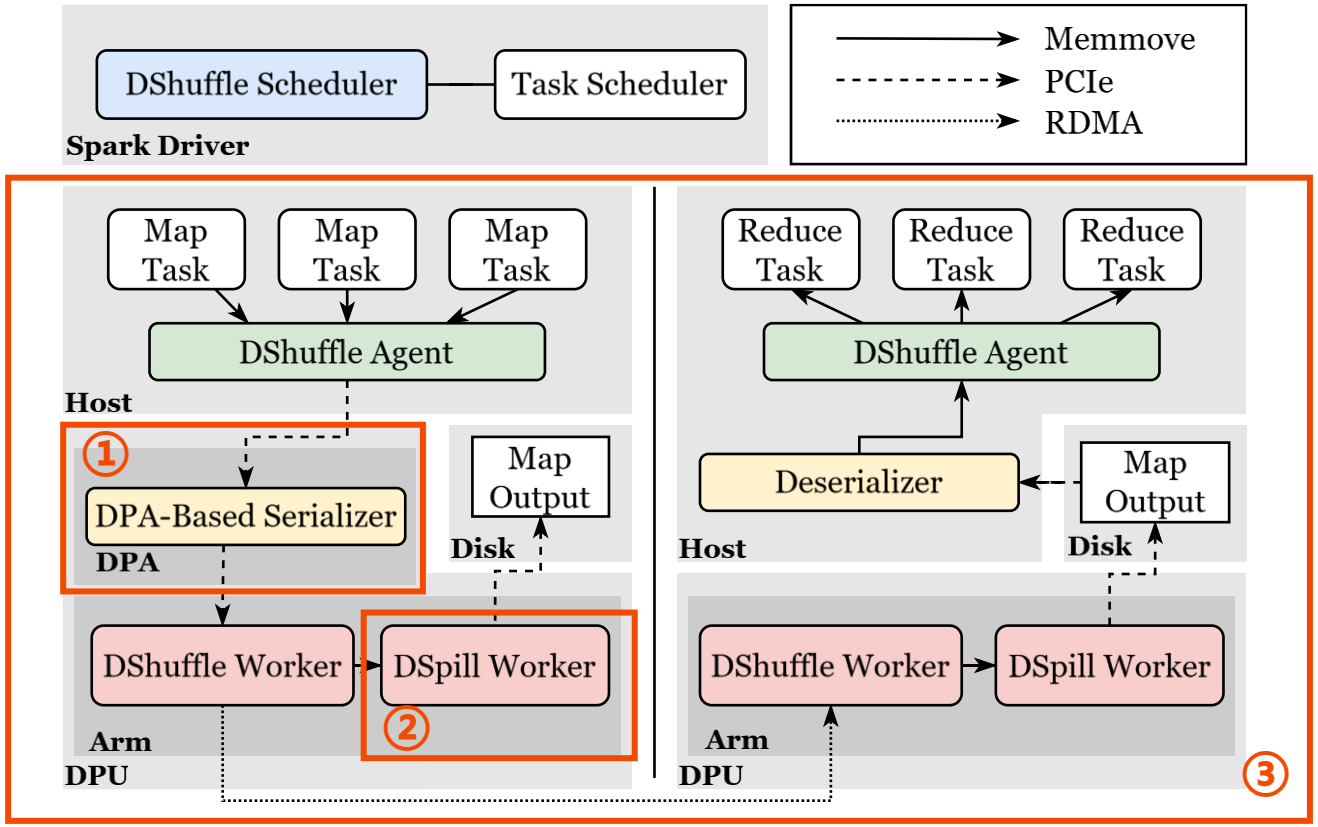
DShuffle aims to completely eliminate the CPU overhead of shuffle operations.




- **Shuffle Agent (Host)**
  - Interact with the host-side Spark
- **DPA-based Serializer (DPA Subsystem)**
  - Serialization accelerator using DPA
- **Shuffle worker pool (Arm Subsystem)**
  - Shuffle worker for executing operators like Agg, sort, ...
  - Spill worker for network and disk I/O
- **Shuffle scheduler (Driver Node)**
  - Detect the DPU status and schedule shuffle tasks.

# DShuffle Overview

DShuffle leverages three key technologies to fully harness the capabilities of DPU.

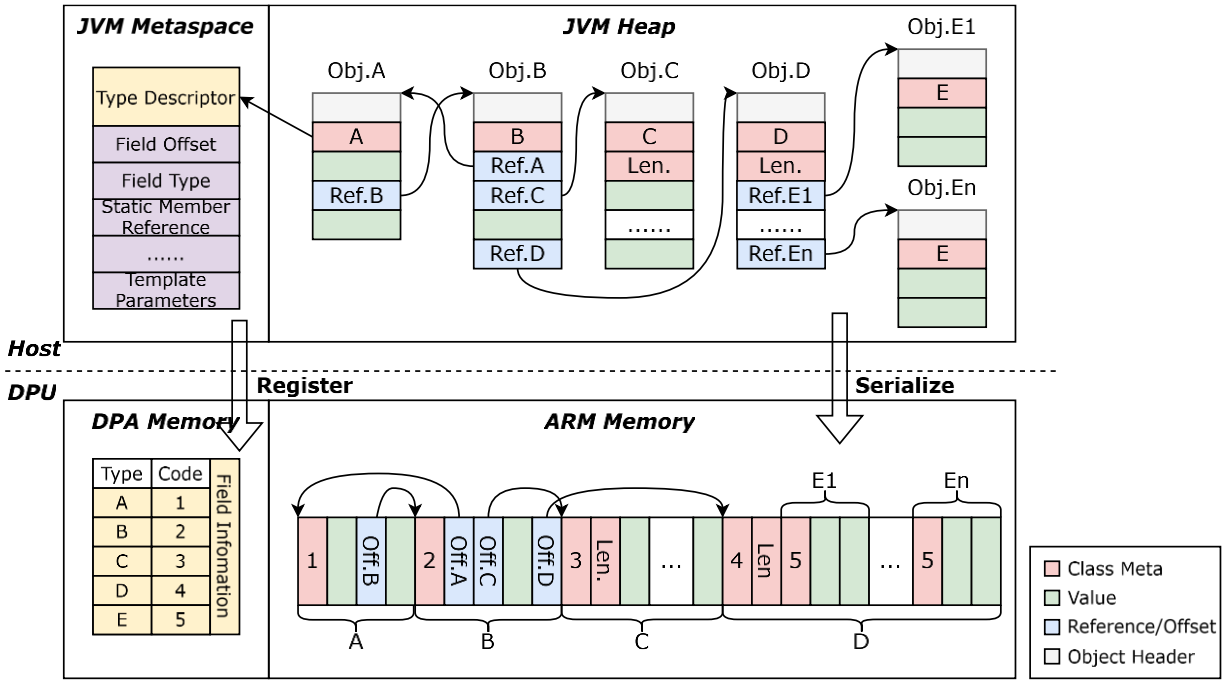



- **Techniques1:** DPA-Accelerated Serialization
- **Techniques2:** DPU-Direct Spilling
- **Techniques3:** Fined-Grained Pipeline Shuffle

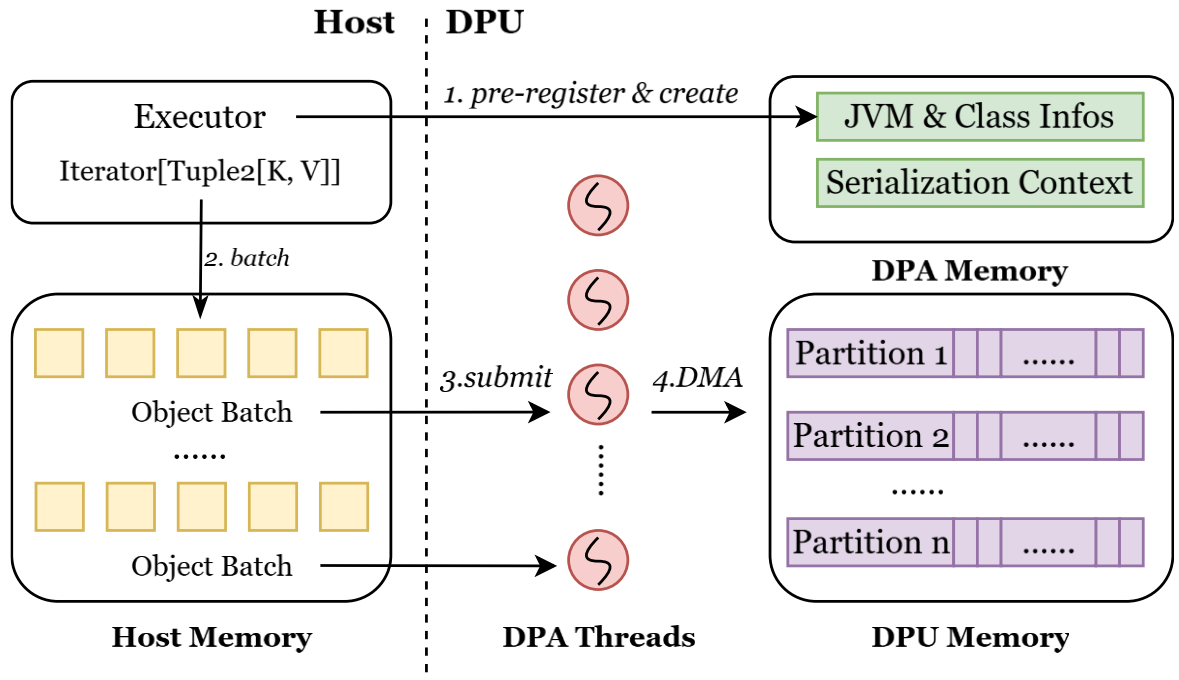
# DPA-Accelerated Serialization

DShuffle serializes Java object arrays in parallel using multiple DPA hardware threads.

## Dedicated serialization format



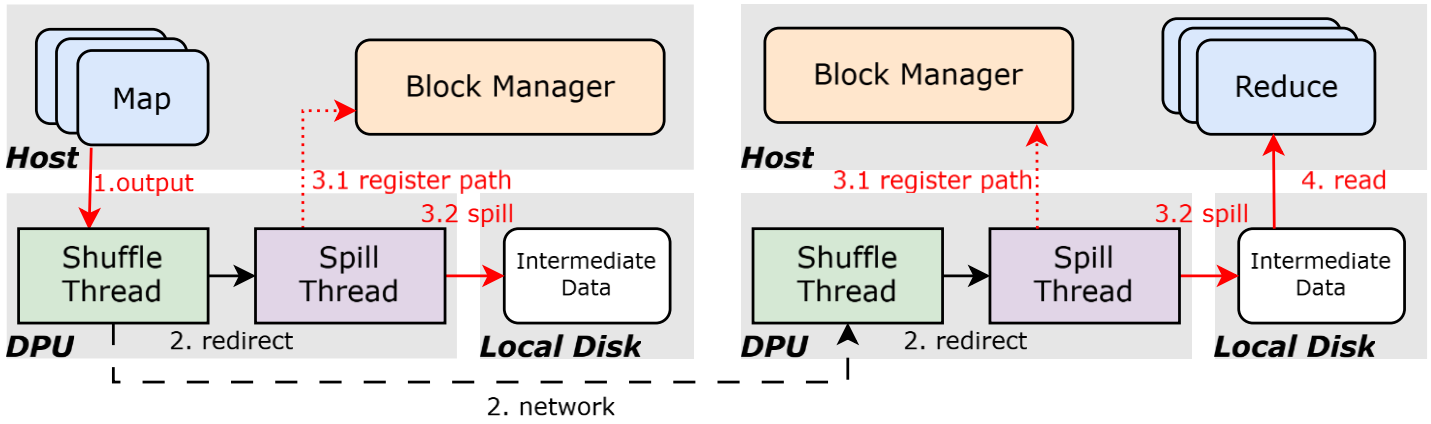

## Parallel serialization

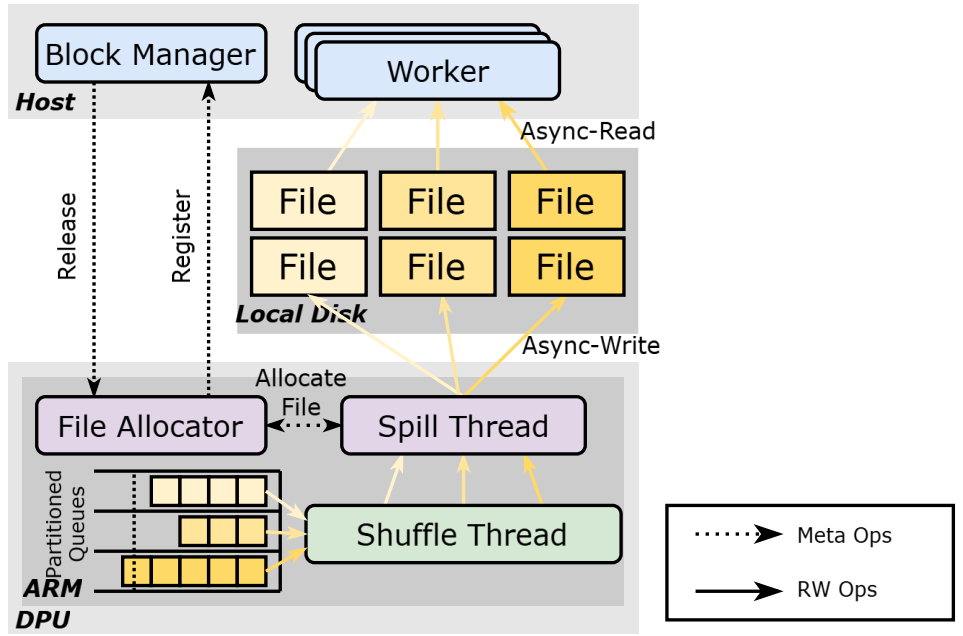



**CPU Overhead ↓ Serialization Performance ↑**

# DPU-Direct Spilling

DShuffle directly writes the intermediate data to the local disk via PCIe P2P transfer.



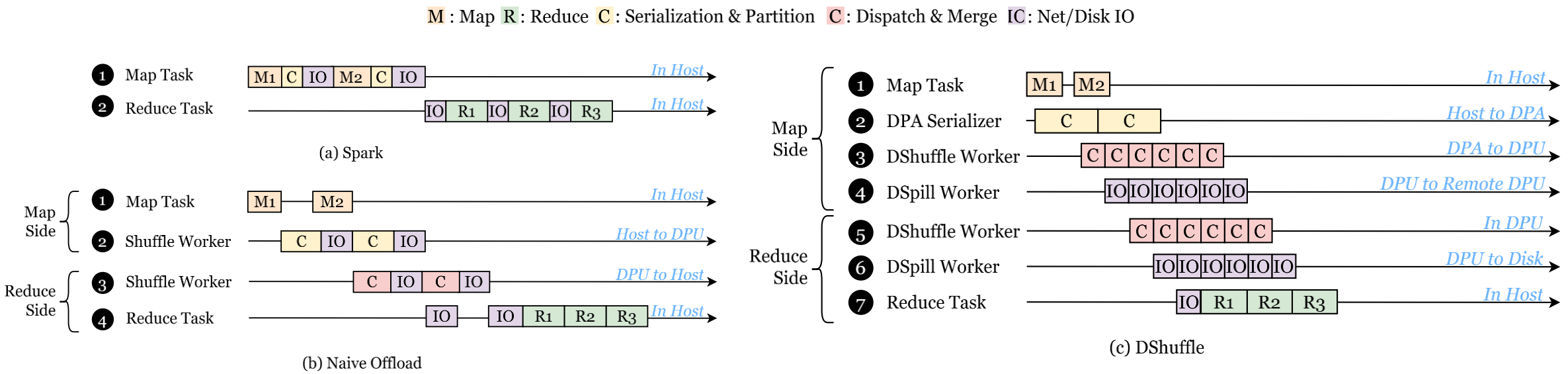




- Pre-register files and mount the host in read-only mode
- Use PCIe P2P to transfer data between DPU and SSD

**Data Copies ↓ GC ↓ CPU Overhead ↓**

# Fined-Grained Pipeline Shuffle

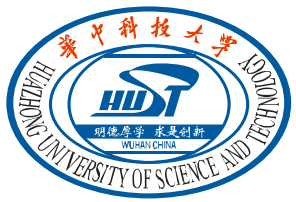
DShuffle builds a finer-grained shuffle pipeline across the host, DPA, Arm cores, and RDMA NIC.




**Shuffle Performance ↑ DPU Utilization ↑**

# Agenda

- Background & Motivation
- Design & Techniques
- Evaluation
- Conclusion




武汉光电国家研究中心
WUHAN NATIONAL LABORATORY FOR OPTOELECTRONICS


# Evaluation Settings

- **TestBed (2 compute nodes, 1 driver node)**

| Hardware | Node Specification |
|---|---|
| CPU | Intel Xeon 6418H 24 cores @ 4.0GHz |
| Memory | 64GB DDR4 |
| SSD | Samsung 980 pro 2TB |
| OS | Ubuntu 22.04 LTS, Linux Kernel 6.8 |
| DPU | NVIDIA BlueField-3 DPU |

- **HiBench Workloads**

| Workload | Data Size | Description |
|---|---|---|
| Sort | 285GB | Sort random strings |
| Terasort | 285GB | Sort 100 bytes key-value strings, which imposes higher shuffle pressure. |
| Repartition | 300GB | Repartition random strings |
| Wordcount | 150GB | Count word frequencies in random strings, with low shuffle overhead. |

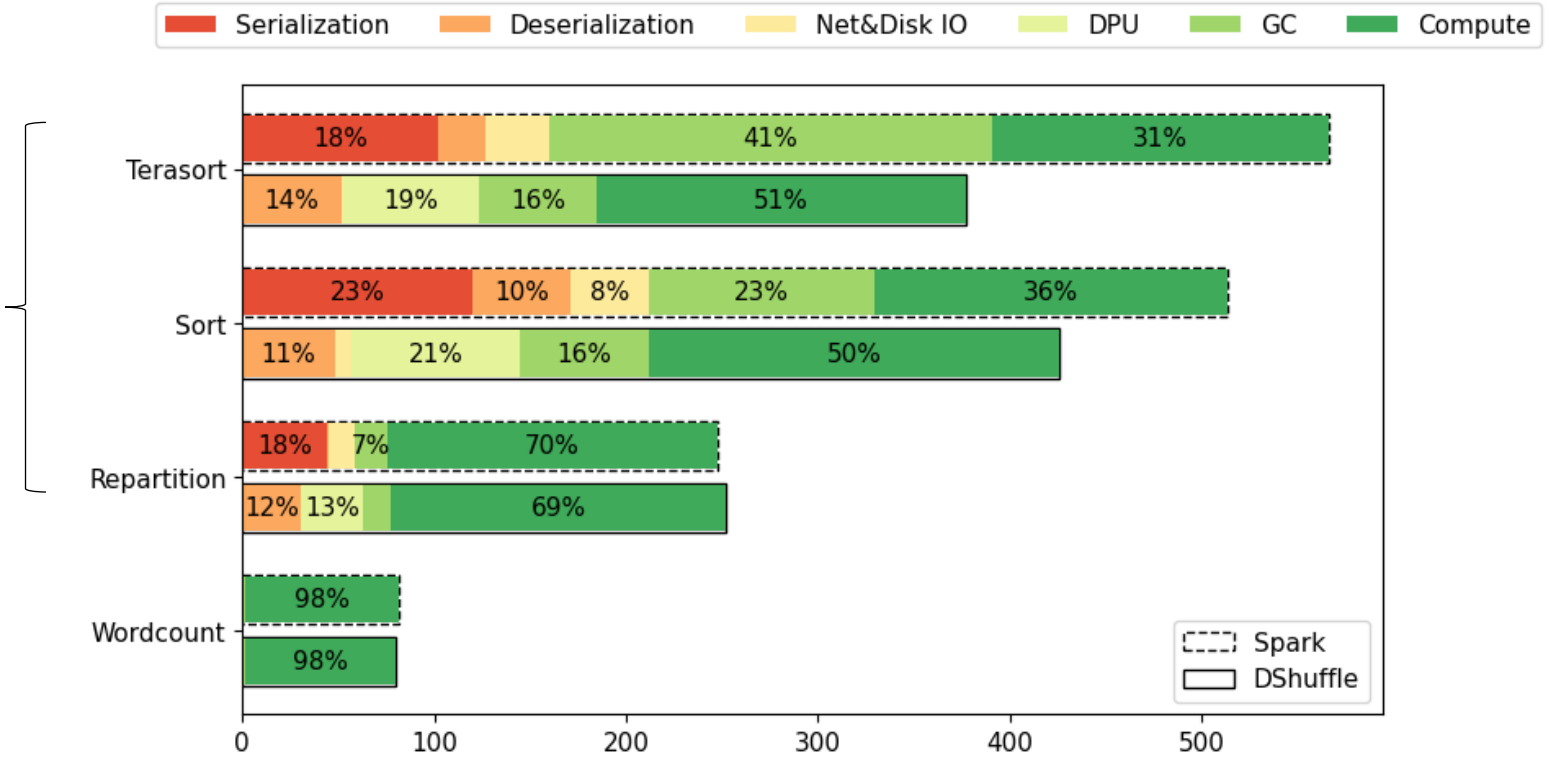
- **Baseline**

| Name | Description |
|---|---|
| Native Spark | Spark 2.4.3 without any offloading, 2 compute nodes with each has a 16C32GB worker instance. |
| Naïve Offload | Similar to SmartShuffle. It offloads shuffle computations and network I/O to DPU, but lefts the serialization and data spilling on host CPU. |
| DShuffle | Fully offloads all shuffle operations to DPU. |

# Performance Improvement

DShuffle improve the end-to-end performance by 14.5% to 33.4%..

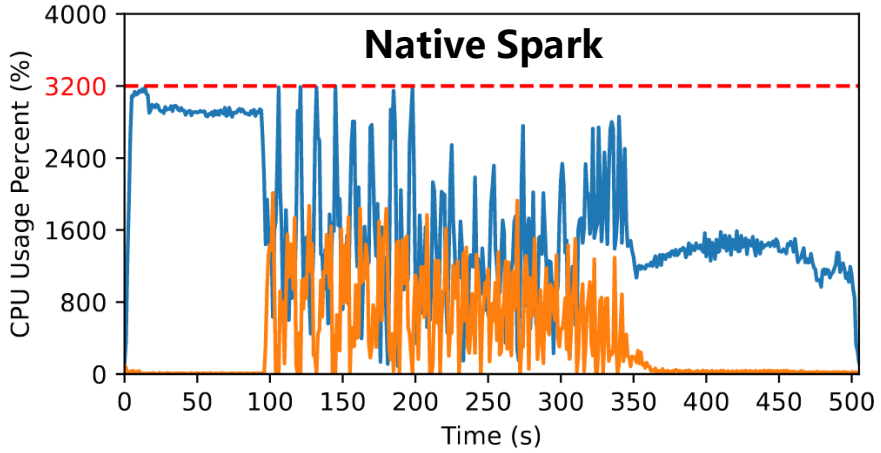
Large volume of intermediate data

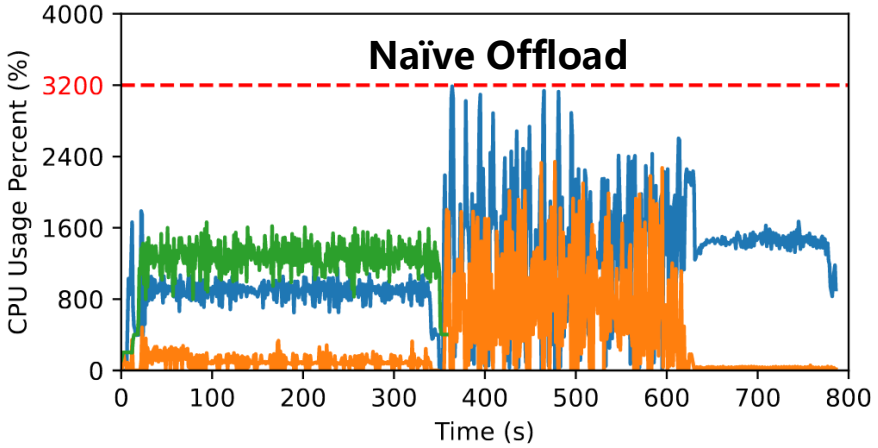


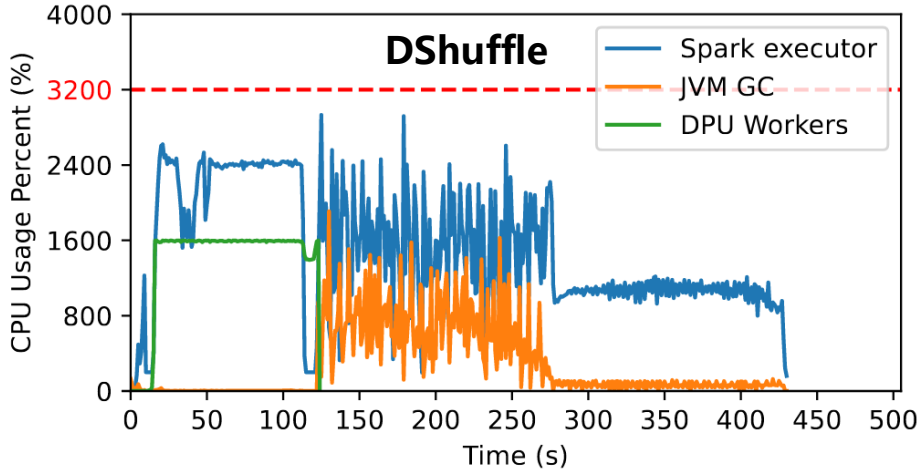

- Network and disk I/O overheads are largely eliminated.
- Serialization and GC overheads are significantly reduced.

# Resource Utilization

DShuffle frees up host CPU resources through offloading.



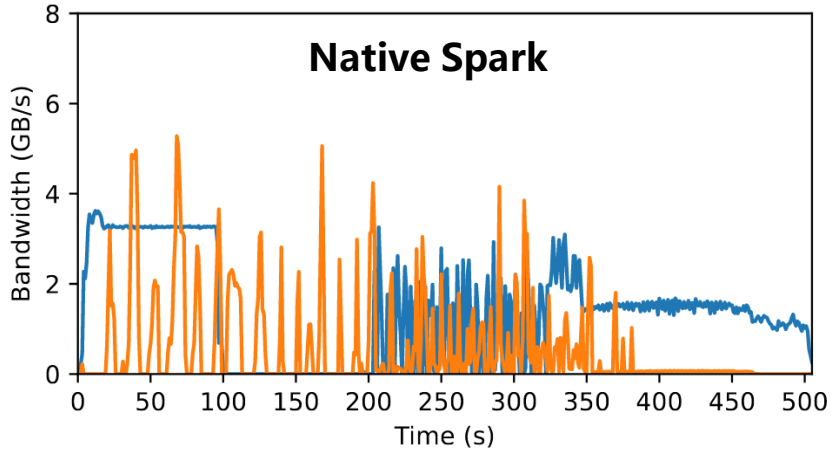



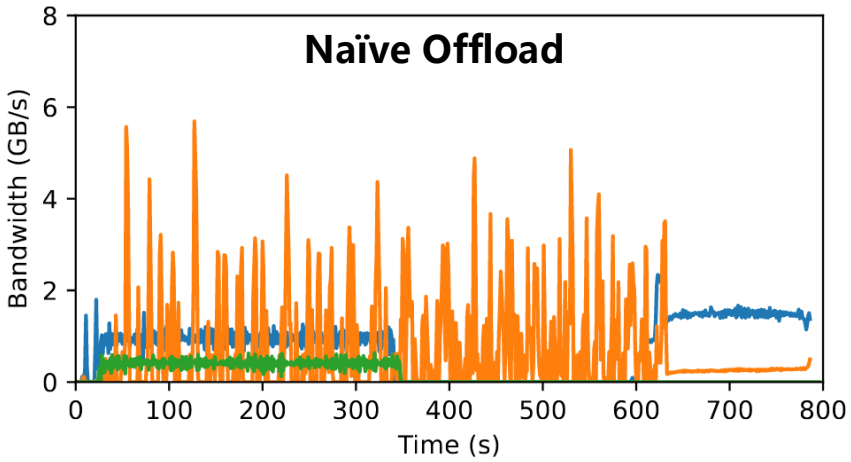



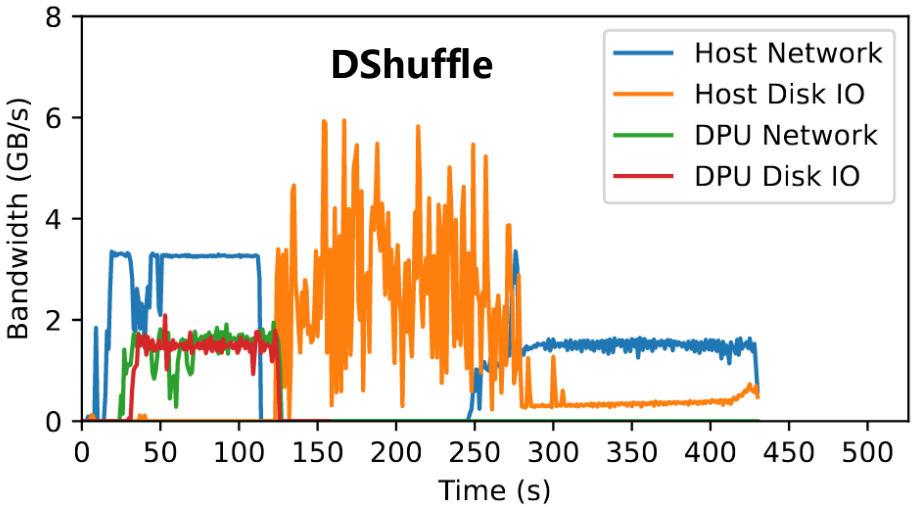

- Map phase: saves around 8 CPU cores, accounting for approximately 25%.
- Reduce phase: CPU consumption of GC is significantly reduced, by approximately 20%.

# Resource Utilization

DShuffle effectively improves the utilization of disk and network devices.



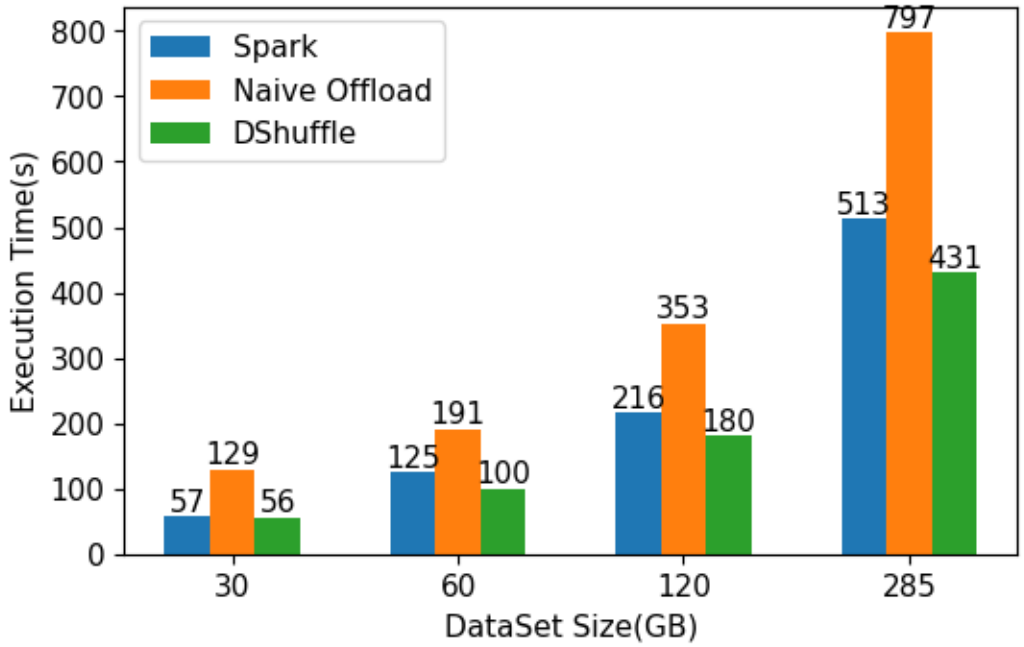



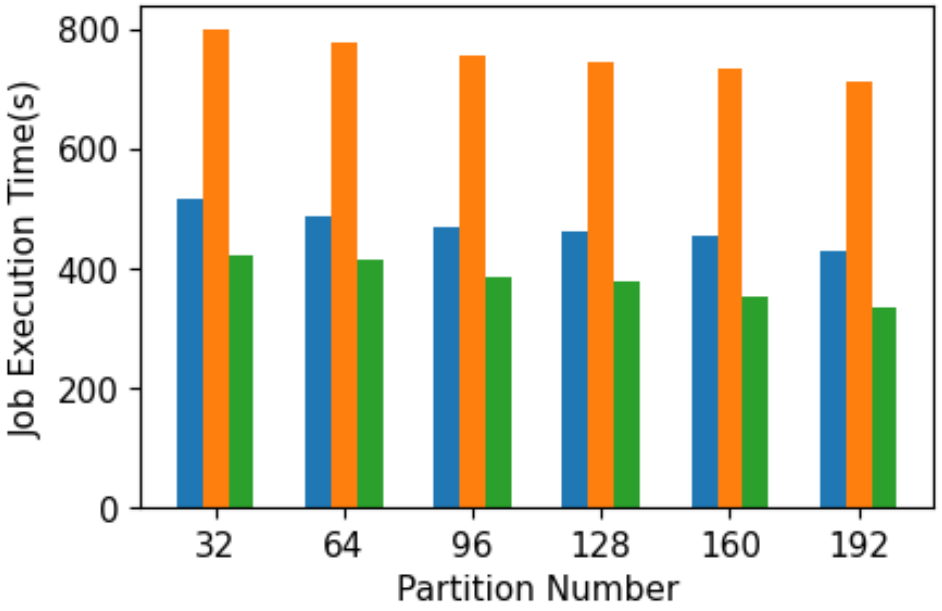




- Map phase：The utilization of disk and network bandwidth is 2× and 3× higher, respectively
- Reduce phase：The host asynchronously and sequentially loads data, achieving up to 90% disk bandwidth utilization.

# Scalability

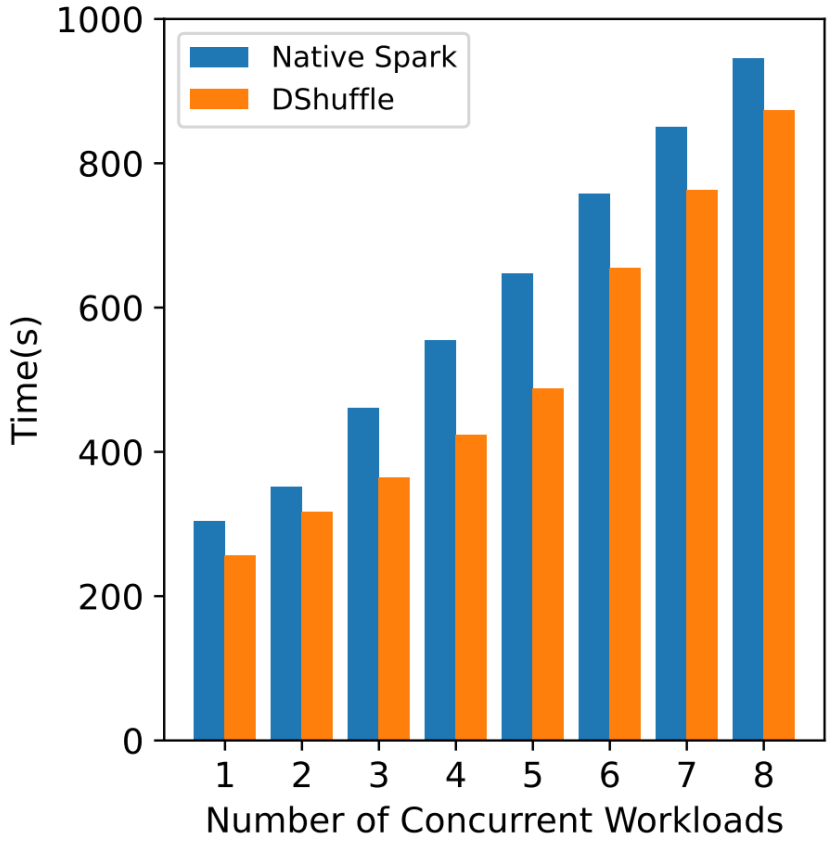
DShuffle delivers consistent performance improvements across different dataset sizes and different number of dataset partitions.

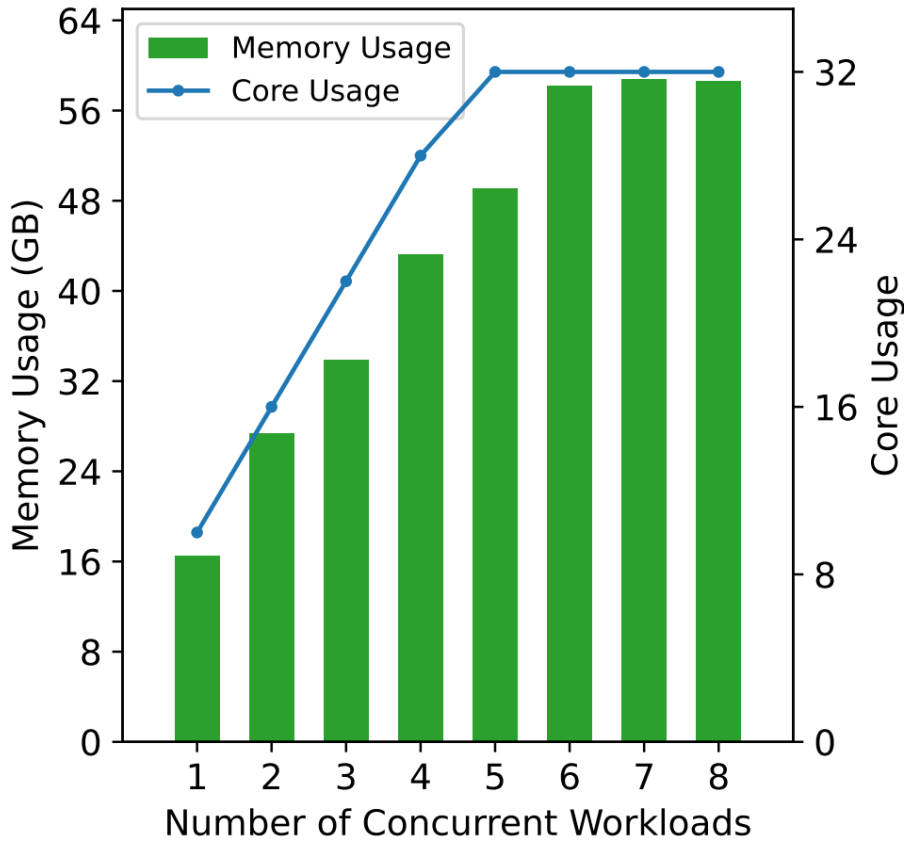


# Scalability

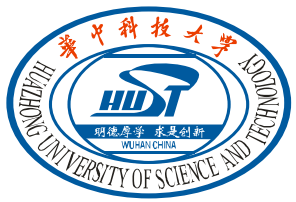
DShuffle exhibits good scalability with the increase of number of concurrent workloads.



# Agenda

- Background & Motivation
- Design & Techniques
- Evaluation
- Conclusion



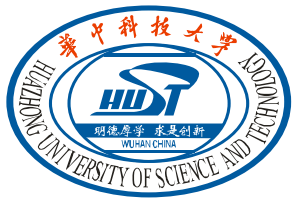

武汉光电国家研究中心
WUHAN NATIONAL LABORATORY FOR OPTOELECTRONICS


# Conclusion

- **Problems**
  - Significant CPU overhead of data shuffle: Serialization, GC, file I/O.
  - Existing optimizations fail to fully eliminate the shuffle overhead.
- **DShuffle designs**
  - DPA-accelerated serialization.
  - DPU-direct spilling.
  - Fine-grained pipeline parallelism.
- **Results**
  - Tests based on Spark and real BF-3 hardware.
  - DShuffle can improve end-to-end performance and completely eliminate shuffle overhead.

# Thanks for attending!
# Q&A

**P**arallel **D**ata **S**torage **L**ab
cding@hust.edu.cn





武汉光电国家研究中心
WUHAN NATIONAL LABORATORY FOR OPTOELECTRONICS